www.icgst.com

Semantic Anti-patterns Detection in UML Models based on Ontology Catalogue

Eman K. Elsayed[1], Kamal A.El-Dahshan[2], Enas E. El-Sharawy[3], Naglaa E.Ghannam[4]
Mathematical and Computer science Dept., Faculty of Science, Al-Azhar University, Cairo, Egypt[1, 2, 3 &4]
emankaram10@azhar.edu.eg[1], dahshan@gmail.com[2], dr_enas_idm@hotmail.com[3], noga.yo99@yahoo.com[4]

Abstract
To develop the pattern quality, we need to clear them from anti-patterns. This paper proposes a general semantic anti-patterns detection tool with an ontology catalogue of anti-patterns information retrieval. The proposed tool is able to identify the anti-pattern and its solution according to the error message. That is to detect the UML class diagram in design level. The system allows the user adding a new anti-pattern at certain category in the catalogue. That is, under the semantic condition to avoid redundancy**.**

Keywords: *Ontology, Anti-patterns, Anti-patterns Catalogues.*

Nomenclature

Symbol	Meaning
UML	Unified Modelling Language
OOS	Object Oriented System
OWL	Web Ontology Language
RDF	Resource Description Framework
MOF	Meta Object Facility
SQL	Structured Query Language
XML	Extensible Markup Language
FOAF	Friend of a Friend
DCMI	Dublin Core Metadata Initiative
FQL	First Order Logic
EA	Enterprise Architecture
OLED	Ontology Lightweight Editor
OCL	Object Constraint Language
VPML	Visual Pattern Modeling Language

1. Introduction
An anti-pattern describes a solution to a problem that generates negative consequences. Each anti-pattern has a name, type, description and solution. They appear in different levels, may be on the code level, design level or in implementation and every level has many numbers of the anti-patterns appear in it. Anti-patterns may be structure anti-patterns, behavioral anti-patterns or semantic anti-patterns, we only concerned with semantic anti-patterns in the Unified Modeling Language (UML) class diagram models.

All types of anti-patterns affect the quality of systems, causing wrong results and making many problems. Detecting anti-patterns is a good step to improve the quality of the models, especially in the design level; it is early more than code level that is considered too late. In addition, in the last years, the number of the anti-patterns became big and it is hard to remember all; there is no general catalogue that collects all types of anti-patterns with solutions of them.

There were many works of anti-patterns detection, as in references [1], [2], [3] and [4] but some of them just detected the inconsistency in UML class diagram and some detected the anti-patterns using different tool but without automatic solutions to the anti-patterns.

In addition, there were many works in the creation of anti-patterns catalogues, as in [5], [6], and [7]. These references created many catalogues, but they accepted the redundancy, the catalogues have no solutions and it covered only certain anti-patterns category. The proposed catalogue covered all anti-patterns categories, gave solutions and did not accept the redundancies. That is by using OWL ontology based to create the proposed catalogue.

In addition, we use the ontology to detect the semantic anti-patterns. An ontology is defined as a formal, explicit specification of a shared conceptualization [8]. An ontology consists of O = {C, P, R, T, I, A}. Where C is a set of concepts or classes; P is a set of properties of the concepts; R is a set of relationships between concepts; T is a set of hierarchically relationships among concepts that is called taxonomy; I is a set of instances and A is a set of axioms.

The ontology provides knowledge about specific domains that the computers and developers can understand. The ontology has many features as semantic properties that will help us recognize semantic anti-patterns and will help to prevent the duplication of the anti-patterns. The semantics of the ontology helps to share information [9]. The important point is the feature of semantic, which is available in the ontology.

The remainder of this paper is organized as follows: Related work is presented in section 2. The anti-patterns detection by ontology is presented in section 3. Then section 4 presented the empirical detection of the semantic anti-patterns. We introduce anti-patterns catalogue in section 5. Section 6 introduces the anti-pattern detection, correction, analysis and elimination. Finally, we present conclusion and future work in section 7.

2. Related Work

There were many works in the anti-patterns detection methods such as:

According to [1], the author detected the anti-patterns in MOF and UML meta-models by using the QVTRelation (QVTR) transformation, which distinguishes between two or more MOF-based models. A pattern is modeled with a Visual Pattern Modeling Language (VPML). This method detected the anti-Patterns through this transformation, but without any automatic suggestions to correct the anti-patterns. Although, this meta-model method have satisfactory performance, but our proposed method gives automatically the number of appearances, the name of these anti-patterns, the way to correct these anti-patterns through OCL constraints, and also detects the UML inconsistency.

In [2], they used an OntoUml editor to detect semantic anti-patterns in OWL but our proposed method is more general, it detects OntoUml anti-patterns in addition to other anti-patterns.

In [3], they presented the inconsistencies detection method in UML class diagram by using Net Beans 7.2.1 IDE. Our proposed method has a simple general way to detect the inconsistencies in addition to other anti-patterns.

In reference [4], the author detected the structure anti-patterns in UML class diagram by creating event-B model and solving the problems. Our proposed method detects the semantic anti-patterns using Ontology and makes a catalogue of all anti-patterns.

In addition, many anti-patterns catalogues were created by different tools and for different reasons.

In reference [5] the author presented an anti-patterns catalogue, this catalogue was just for showing meta-modeling anti-patterns, he described just UML anti-patterns, did not describe any other anti-patterns, giving us just the name of them and the query to detect each one. The catalogue didn't present any anti-pattern solution or the message appears in the case of it existed to be easy for the user to recognize it, so this catalogue was especially for showing meta-modeling anti-patterns not general, no solutions and no error messages. Our work is general as it shows all the anti-patterns and in the case of the user has the error message and does not know the anti-pattern, he gets all information about the anti-pattern that caused this message in an automatic way.

In reference [6] the catalogue showed and described many types of the anti-patterns; it described everyone, its type, its solution and some examples of it. When the user wants to know information about an error message, he cannot find the related anti-patterns. In addition, this catalogue has anti-patterns with different names -as "Blob" & "God Class"- and the same descriptions. Our work saves the time for all this, the user just enters the error message, he gets the anti-pattern information, and in addition, our catalogue prevents the duplicating of the anti-patterns.

In [7] the catalogue presented a list of different anti-patterns, it just described them, but there was no solution to any one of them, and it was for a certain type. Our work is general, gives a description and a solution to every anti-pattern and gets the information of the anti-patterns from the error message versa.

3. Anti-patterns Detection using OWL Ontology Based

Current approaches for identification of anti-patterns operate either at the code level (for software re-engineering purposes) or at the design level (for design quality improvement purposes). Detecting anti-patterns at the design level allows the designer to anticipate the problems that could be generated by an implementation. Detection of anti-patterns at the code level is considered too late. Therefore, we are interested in anti-patterns detection at the design level. There are several metrics approaches to detect anti-patterns as (Coupling, cohesion, complexity and inheritance).

In this paper, we concentrate on the object-oriented design, where we propose a metric-based approach for anti-patterns detection on UML class diagram designs. The proposed method for detecting anti-patterns based on mathematical metric between UML model and other formal models.

We use an ontology to detect the semantic anti-patterns in UML models to get a pattern. As Ontology has many benefits where it

- Improves the quality of the design and software systems.
- Selects semantic access and guided discovery of knowledge in Knowledge engineering and Management.
- Prevents the duplication.
- Ontology has the reasoner that we can use to check ontology consistency that is helping to avoid some anti-patterns.
- Ontology has many plugins as prompt plugin that helps to merge ontologies and make information retrieval from one project after the merging or any other processes it does.
- Ontology is the backbone of the semantic, so almost any semantic information will be stored in the form of it and published in the form of one of its languages as RDF or OWL.


- Ontology can perform a specific purpose as "Thesaurus" in the field of information retrieval or a model represented as OWL in the field of linked-data or XML schema in the context of databases or in software development.

We mapped UML to OWL, as there exists a semantic correspondence between them, which allows the automatic translation or conversion from UML to OWL. In addition, Ontology and UML are both object oriented models, so they have similar meanings as in table1.

Table 1. Correspondence between UML and Ontology

UML	**Ontology**
Package name	Namespace
Class	Class
Instance	Individual
Attributes	Properties
Multiplicity	Cardinality
Generalization	Subclass of (Hierarchy)
Association	Relation
OCL constraints	Axioms containing these rules
(Superclass, Subclass)	The hierarchy concept (Taxonomy)

The first step when we want to create the ontology is identifying all classes in the domain. Concepts refer to classes that are the most important component in ontology, where a class is a set of individuals that share common properties. According to [9] OWL classifies everything in terms of semantics that is helping the machine reader to know if an individual is a member of a class and its object properties or data properties related to that OWL class or not. Ontology describes the concepts and the relationships that are important in a particular domain. That is to provide a vocabulary for the domain [10]. A Class may have subclasses, or maybe it is a subclass of another class, but all of them are subclasses of the class "Thing" that is the root class in owl. The class hierarchy is called taxonomy.

In addition, a class may have some instances (individuals) and may not, it is not necessary to have.

Ontology has three types of properties, which are

- Object properties: Are the relationship between one individual and another individual of two owl classes.
- Datatype properties: Are the relationship between instances of classes.
- Annotation properties: They are used to add information to classes, individuals, and the others type properties.

Figure 1 presents our proposed system from the beginning, when any user enters into our system; he does one of three. First, he may has UML model and want to detect the semantic anti-patterns in it, second he may has an error message and wants to know the anti-pattern description and solution, or finally he may has new anti-pattern and wants to add it to the catalogue. The output will be the detected anti-patterns for the first choice, or anti-pattern information for the second choice, or the inserted anti-pattern information for the third choice.

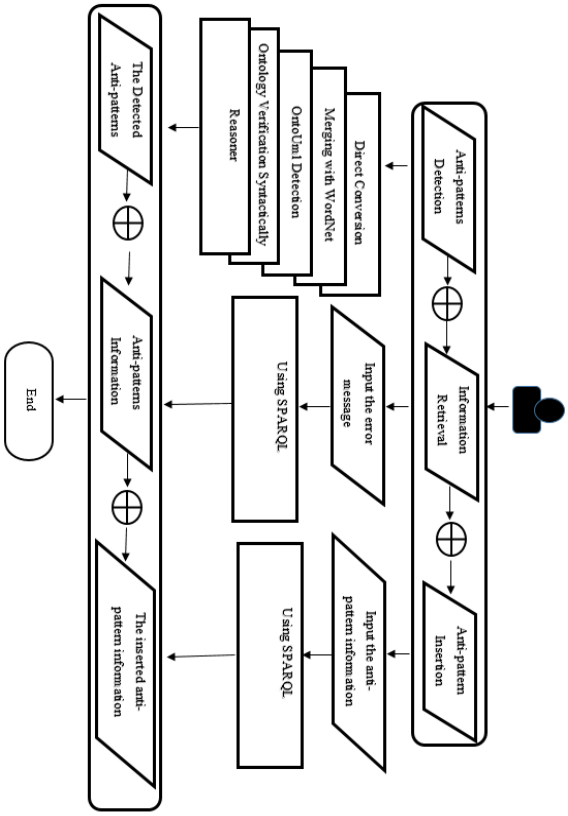


Figure 1. The proposed system

The following is the pseudo code for our system which applyes in more details in the following section.

Input the choice
***Case** (Anti-patterns Detection)*
Insert UML model
Convert the UML to OWL
Run the detection processes
Print the detected anti-patterns list
***Case** (Information Retrieval)*
Input the error message
Run semantic search SPARQL query
Print the anti-pattern information
***Case** (Inserting new anti-pattern information)*
Run inserting SPARQL query
Output the new version of Anti-patterns ontology
End

4. Empirically Detecting Semantic Anti-Patterns

We use the Protégé platform [11] which is an open source tool of ontology to do our work. We study the identification and repeating of these anti-patterns across different domains, different sizes and complexity.

We study, nine UML class diagrams which we uploaded them as UML templates to be used as patterns. These patterns are in (ATM UML class diagram [12], Library and Android UML class diagrams [13], Hasp UML class diagram [14], Seminar, Order and Auction UML class diagrams [15], SWT UML class diagram [16] and Furniture UML class diagram [17]).

To illustrate our work we use the example presented in Figure 2. The Hospital UML class diagram contains six classes "Hospital, Booking, Doctor, Patient, Continuous and not Continuous ". Class "Booking" has five attributes and three operations, which has one booking for everyone patient. Class "Doctor" has five attributes and no operations that will treat many patients. Class "Patient" has no attributes and four operations and the patient may be continuous or not continuous patient. Class "Hospital" has two attributes and no operations, class "Continuous" has two attributes and no operations and finally, class "not Continuous" has two attributes and no operations. The model also has seven associations, some of them with known multiplicity and some are not.

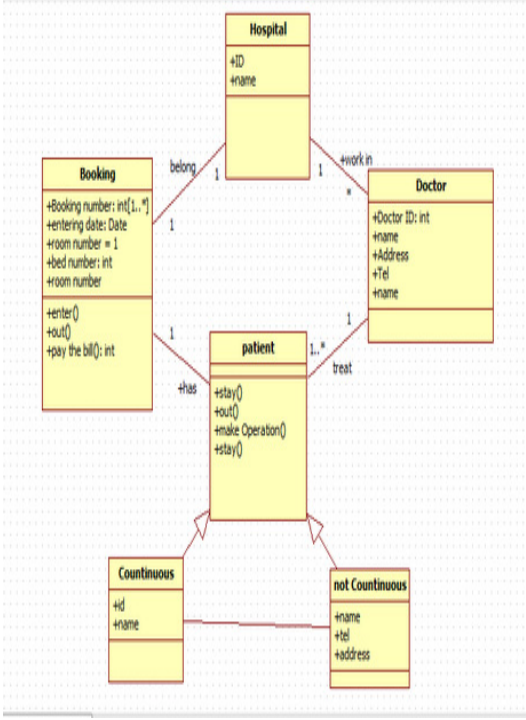


Figure 2. Hospital UML class diagram

For our example in Figure 2, when we transformed it to OWL directly:

- We detected seven anti-patterns, which are (Class has no operations, Class has no attributes, Class has no attributes and no operations, An attribute has no multiplicity, An attribute has no initial value, An operation has no return type and An association multiplicity is ambiguous). As classes "Doctor, Continuous and not Continuous" have no operations, class "patient" has no attributes; attribute "address" in class "Doctor" has no multiplicity. The same attribute "address" also has no initial value as in Figure 3. So Figure 3 shows both anti-patterns (Attribute has no multiplicity and attribute has no initial value). Class "Booking" has operation "Pay the bill" with a return type "Integer", but the operation "stay" in class "patient" has no return type , this anti-pattern is happening for all the rest of operations in class "Patient", and for operations "enter, out" in class "Booking". Finally, there are seven associations, three of them have associations with known multiplicity, but the association connects classes "Booking" and "Patient" has two ends, one with name "Has" has no multiplicity and another with multiplicity.

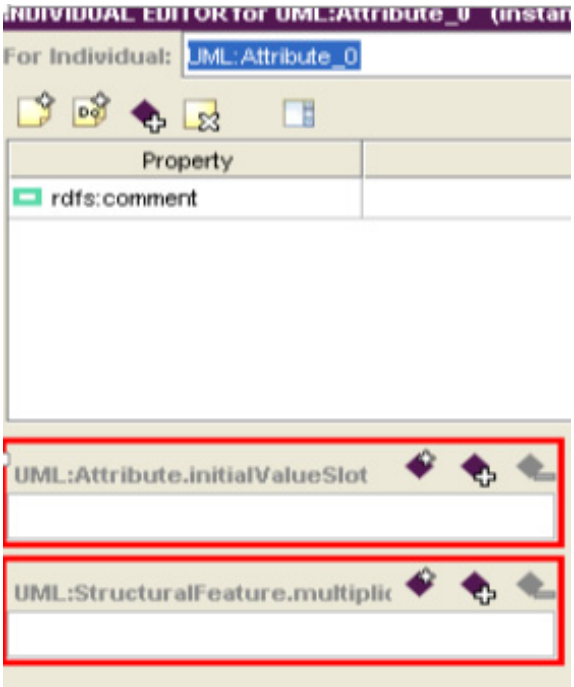


Figure 3. An attribute has no initial value and no multiplicity

- We detected "Same name" anti-pattern through merging the converted OWL ontology of UML with WordNet as used in [18], WordNet measures the similarity of meaning between two strings. But it is not enough. That is because there are semantic anti-patterns

were not detected by using WordNet or any linguistic ontology exactly in the UML model. In our example, the Hospital UML class diagram, this anti-pattern was detected three times, class (Booking) has attributes with the same name "room number", class "Doctor" has attribute "name" repeated twice and finally, class "patient" has an operation "stay" twice also.

- We also transformed UML to OWL not directly, by transforming the UML to OLED file in OntoUml editor, which is an Ontology Lightweight Editor (OLED) [19]. It imports the (Enterprise Architecture) EA file, which is created or imported from UML models. OntoUml editor has a list of twenty-one semantic anti-patterns giving us the name and the number of appearances. Some of them are explained in [2]. In our example, we detected three anti-patterns through using the validation of OWL anti-patterns in OntoUml editor. The three anti-patterns are "Association Cycle" as we have a cyclic relation among classes "Hospital, Patient, Booking and Doctor". The second anti-pattern is "Imprecise Abstraction" as we have a class "Patient" with upper multiplicity greater than one has two subtypes and it has a relation with the class "Doctor". The third anti-pattern is "Relation Composition" as there is an association between classes "not Continuous" and "Continuous" that is supposed to be disjoint classes as in Figure 4.

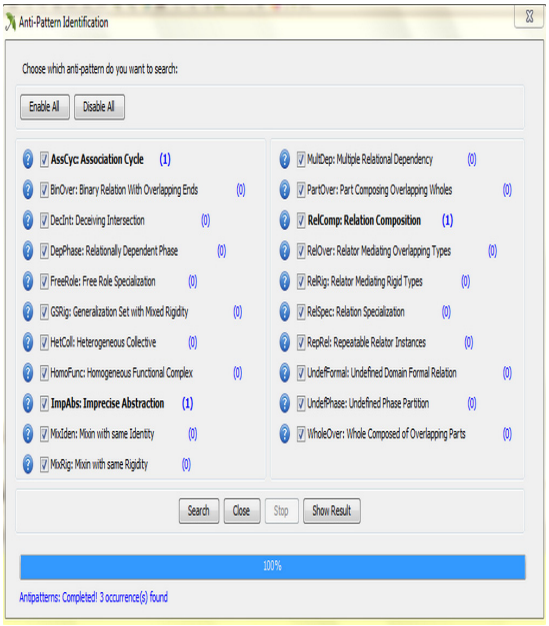
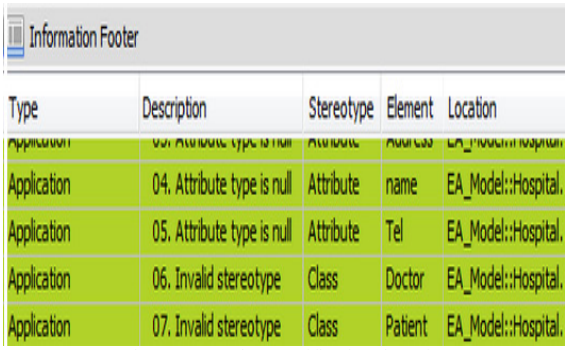


Figure 4. OntoUml semantic anti-patterns detection

In addition, by OntoUml we can detect the syntactical anti-patterns not just the semantic anti-patterns through using Ontology verification syntactically, it detects the anti-patterns (attribute has no data type, the class multiplicity equal zero and Invalid stereotype). In our example "Hospital" UML model, this verification detected the anti-patterns "Attribute has no data type" eight times, the anti-pattern "Class multiplicity equal zero "one time and "Invalid stereotype" nine times as in Figure 5.



Type	Description	Stereotype	Element	Location
Application	04. Attribute type is null	Attribute	name	EA_Model::Hospital.
Application	05. Attribute type is null	Attribute	Tel	EA_Model::Hospital.
Application	06. Invalid stereotype	Class	Doctor	EA_Model::Hospital.
Application	07. Invalid stereotype	Class	Patient	EA_Model::Hospital.

Figure 5. Ontology verification detection

- By the reasoner of ontology, we detect the inconsistency anti-patterns. This Reasoner detection can be done in the direct or not direct transformation. According to [3], the class diagram inconsistencies are (Similar name, Generalization and Disjoint, Multiplicity constraints and Cyclic Inheritance). ". After OntoUml validation and ontology verification, we transform the OLED file to OWL file. This transformation gives us the chance to detect the inconsistency using the "reasoner". In the "Hospital" UML model, "reasoner" detected three anti-patterns "same name", which we explained in the anti-patterns detected by merging with WordNet, "Cyclic Inheritance" which we explained in the anti-patterns detected by OntoUml, and "Generalization and Disjoints" which we explained in the anti-pattern detected by OntoUml with name "Relation Composition" between classes "Continuous and "not continuous".

Generally, according to this detection method, we detect thirty-six anti-patterns, which are thirty-three different semantic anti-patterns and three syntactical anti-patterns. As direct conversion detects seven anti-patterns, OntoUml has a list of 21 semantic anti-patterns, WordNet has one anti-pattern and Reasoner have four anti-patterns. All anti-patterns are in table 2, every detection tool has a true sign of its anti-patterns, some of the anti-patterns can be detected by more than one detection tool as Reasoner and OntoUml detect the anti-pattern "Association Cycle". And Reasoner and WordNet detect "Same name", so thirty-four anti-patterns will be just presented in table 2.

Table 2. The anti-patterns detected

	The anti-pattern	Direct Conversion	OntoUml	WordNet	Reasoner	Proposed way
1	Class has no attributes	√				√
2	Class has no operations	√				√
3	Class has no attributes and no operations	√				√
4	Attribute has no multiplicity	√				√
5	Attribute has no Initial value	√				√
6	Operation has no return type	√				√
7	Association multiplicity is ambiguous	√				√
8	Same name			√	√	√
9	Generalization and Disjointness				√	√
10	Multiplicity constraints				√	√
11	Association Cycle		√		√	√
12	Relation Specialization (RS)		√			√
13	Relation Between Overlapping Subtypes (RBOS)		√			√
14	Mixin with same Rigidity		√			√
15	Binary relation with overlapping Ends		√			√
16	Deceiving Intersection		√			√
17	Relationally Dependent phase		√			√
18	Free role specialization		√			√
19	Generalization set with Mixed Rigidity		√			√
20	Heterogeneous Collective		√			√
21	Homogeneous Functional Complex		√			√
22	Imprecise Abstraction		√			√
23	Mixin with same Identity		√			√
24	Multiple relational Dependency		√			√
25	Part Composing overlapping whole		√			√
26	Relation composition		√			√
27	Relator Mediating Rigid Types		√			√
28	Repeatable relator Instances		√			√
29	Undefined Domain Formal Relation		√			√
30	Undefined phase partition		√			√
31	Whole Composed of Overlapping Parts		√			√
32	Attribute has no datatype		√			√
33	Class multiplicity equal zero		√			√
34	Invalid Stereotype		√			√

In the study of the nine UML patterns plus our example, we divide the detected anti-patterns into four groups, which are (semantic anti-patterns of attributes, semantic anti-patterns of class, semantic anti-patterns of operations and semantic anti-patterns of association) as represented in table 3.

Table 3. Occurrences of Semantic Anti-patterns in the UML patterns

The anti-pattern group	**% of occurrences across models**	**Total # of occurrences**
Semantic anti-patterns of attributes	%35.7	158
Semantic anti-patterns of class	%42.1	186
Semantic anti-patterns of operations	%5.4	24
Semantic anti-patterns of association	%16.7	74
Total		442

Figure 6 shows a snapshot of the system's output in the case of the detection process. The screen shows all anti-patterns were detected by our proposed method. But the detected anti-patterns in our "Hospital" example only have the true sign "√ " and if the user wants to retrieve the information about the detected anti-patterns, he just clicks the "Descripition and Solution" button. Figure 14 and Figure 15 display the example of the query output in this case

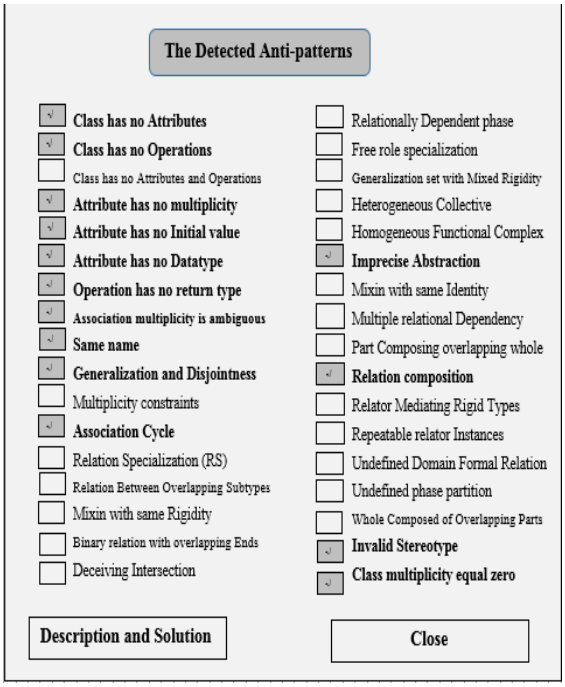


Figure 6. The output of the detection process

5. The proposed Catalogue of Anti patterns

There are many anti-patterns catalogues, but most of them just focuses on one special aspect. Many catalogues just present the anti-patterns that were detected by the work of any developer. In some cases, we cannot find a description of the anti-pattern, why it happened or the error message of it. We cannot find all information of the anti-pattern as they are distributed. We need an easy way to group and describe all the anti-patterns, helping to understand them to be avoided and to be easy for any user to get information that he needs. In our work saving the time, the money and getting correct results. Our proposed catalogue collects all the anti-patterns of different types such as design anti-patterns (semantic and structure), code anti-patterns, implementing anti-patterns, etc. As all the anti-patterns share the same domain that is the word (Anti-pattern) and all different types of the anti-patterns are concepts in this domain. Therefore, it was likely for us to use the ontology with its features specially the semantic feature for helping in semantic information retrieval.

We use the ontology editor protégé to create the catalogue. In addition, we can use the ontology reengineering tools as merging tool "as prompt" to

merge recent catalogues without duplicates. Then we use the reasoner to check the consistency of the catalogue.

We created the class "Anti-patterns" under the protégé root class "Thing". For every anti-pattern class, we specify its annotations, descriptions and its properties; we add possible error messages for every anti-patterns as individuals from different sources. According to [6] the anti-patterns may be in more than one category, which are (Software Architecture anti-patterns, Software Development anti-patterns, Software Management anti-patterns, User Interface anti-patterns and Organization anti-patterns). Every category has its own anti-patterns of different types and different detecting tools. The Software development anti-patterns were classified into (Design anti-patterns, Code anti-patterns, Scoping anti-patterns, UI anti-patterns… etc.) [20]. UML anti-patterns may be semantic or structure anti-patterns, also both of them has its detection tools, so in the catalogue, we create two classes, one of the semantic anti-patterns containing some tools that detect these anti-patterns as subclasses, each tool has the anti-patterns were detected by it. The second class "the structure anti-patterns" contains some detection tools and the anti-patterns that were detected by them. The anti-patterns ontology catalogue is shown in Figure 7.

Figure7. Anti-patterns Ontology catalogue

This was for UML class diagram, but the same thing is also available for all types of UML diagrams, not just this, but also for any other modelling language, programming language like Java, C++, etc. This is just a case study, meaning that our catalogue is general; it has the ability to collect all anti-patterns. So anyone can add any type of anti-patterns to the catalogue, just adds all the information about it to make it easy for any other user to recognize it. First, we ensure that the anti-pattern is not already in the catalogue with a different name. In addition to the creation of the catalogue, we proposed a simple way to retrieve any anti-pattern information by using SPARQL. If the user has an error message, but he does not know the reason that causes this message, he simply enters the message to our system and all anti-pattern information will be the query result. This query result contains the name, type, detecting tool and solution to the anti-pattern. That is mean that all information are grouped in the same place.

Many users do not know many anti-patterns, and why the error message appears, almost they do not know the reason, so they just stop to search more and more in different sources of information, reading many anti-patterns descriptions, trying to reach for the anti-pattern to know the reason of this message. The proposed way helps them for saving the effort and the time by logging into the catalogue, writing the error message and in an automatic way, all information of the anti-pattern will appear. Moreover, in the case of the error message is not exist, the query result will be the anti-patterns names that their error messages have word(s) of the user message. Figure 8 presents the screenshot for structure anti-pattern that was detected by Rodin platform [21]; this anti-pattern is the "Invalid Identifier" anti-pattern. In addition, Figure 9 presents the SPARQL query of the anti-pattern; this query contains the complete error message not just a word. Figure 10 presents the query result.

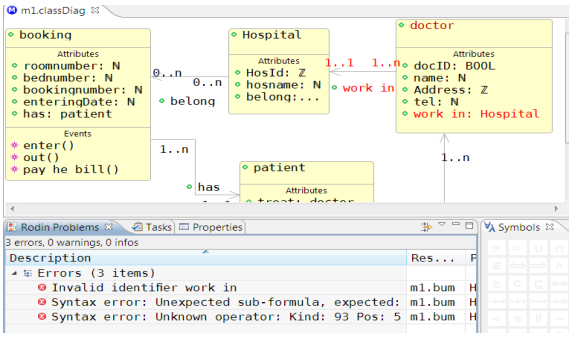


Figure.8. Invalid identifier anti-pattern in Rodin

In the case we have, the error message is not complete or is not the same according to the soft we use; the query result will be the more close anti-patterns to the message we enter. Figure 11 and Figure 12 are two anti-patterns that have a shared word as "attribute" in their error messages, when we query using this word; the anti-patterns that the error message of them contains this word will be the query result as in Figure 13.

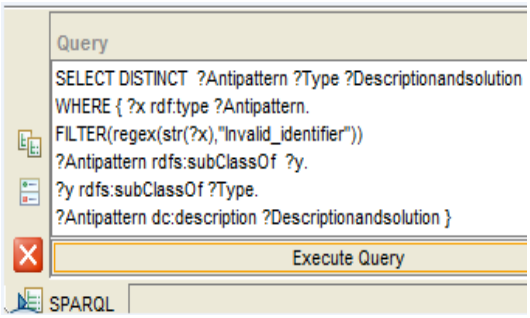


Figure9. Invalid Identifier query

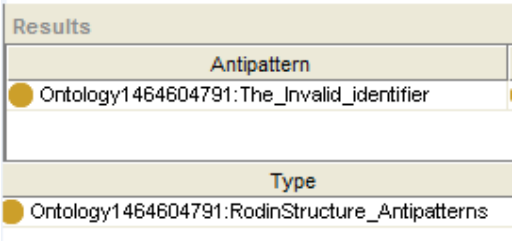


Figure 10. Invalid Identifier query result

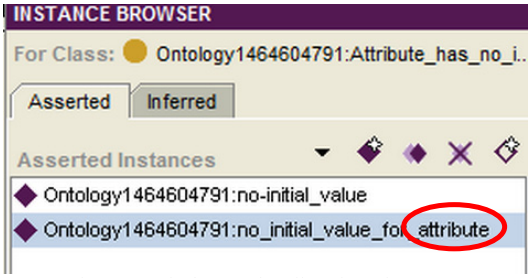


Figure 11. The keyword attribute in anti-pattern1

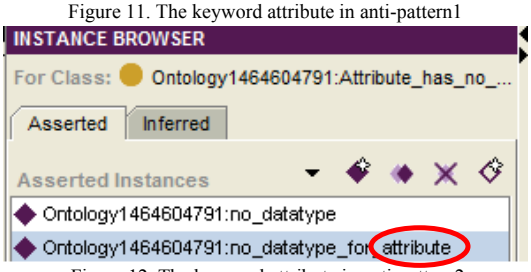


Figure 12. The keyword attribute in anti-pattern2

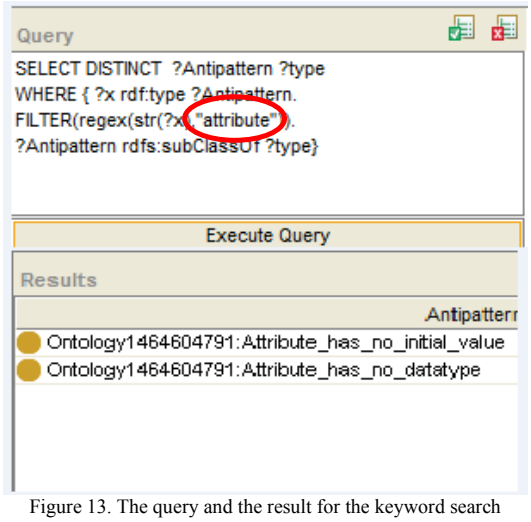


Figure 13. The query and the result for the keyword search

For the output of the detection process, when the user wants to know the description and the solution of the anti-patterns, the system retrieves the information that's stored in the catalogue using the query in Figure 14 and its result in Figure 15.

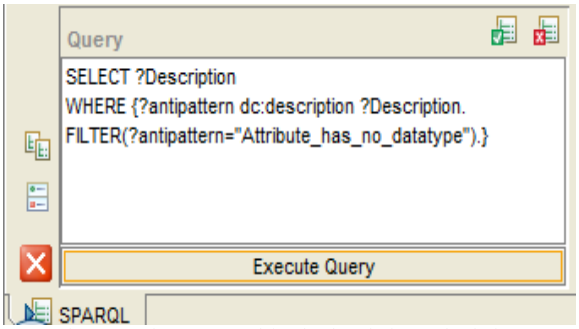


Figure14. The query used for the description and solution

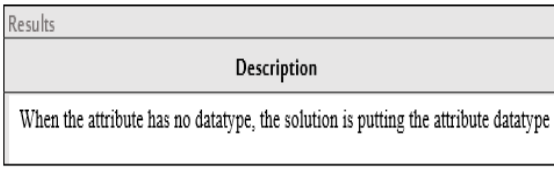


Figure15. The query result

Figure 16 shows the snapshot for the anti-pattern insertion in the catalogue which is for the user to enter the anti-pattern information.

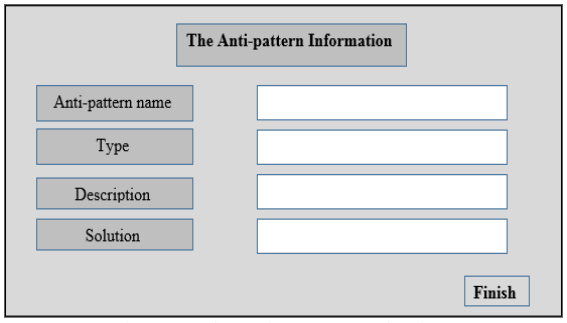


Figure 16. The Anti-pattern Insertion form

6. Anti-Patterns Detection, Correction, Analysis and Elimination

In this research work we use more than one detection tool, one of them was OntoUml that allows the automatic detection of anti-patterns and offers semiautomatic correction through the automatic generation of OCL constraints as in Figure 17 and Figure 18. The anti-pattern "Relation Composition" in the association between the classes "Patient" and "Doctor" is corrected by choosing the refactor option through answering some questions, after that the OCL constraints are generated to correct the anti-patterns in an automatic way.

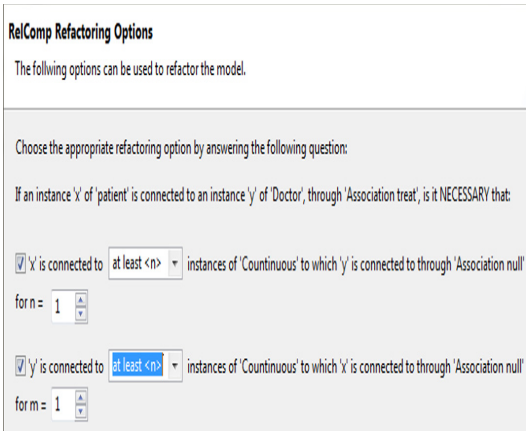


Figure 17. Choosing the refactor option

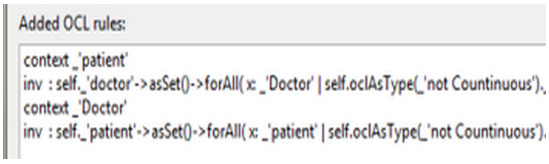


Figure 18. Automatically generated OCL solutions to the anti-pattern

In the direct conversion between UML and Ontology, we detect some anti-patterns that can be corrected through putting the missed part in Ontology directly, and then the anti-pattern will be corrected as in Figure 19. We correct the anti-pattern "Attribute has no initial value" through the sign "+", we add an initial value to the attribute in Ontology directly.

Figure 19. Correction in the direct conversion

For the "same name" anti-pattern which we detect by WordNet, the user just changes the name of the element.
For the inconsistency anti-patterns, the user sees the reasoner result and changes according to the result.
In the anti-pattern catalogue, we enter every anti-pattern with its solution into the catalogue. Therefor at the information retrieval process, each query result contains the anti-pattern associated with its solution.
The pseudo code for this step is as follows:
read the detected anti-patterns
IF(The anti-patterns are in the OntoUml list)
then
follow OCL constraints
Else IF (The anti-patterns are in direct conversion)
THEN
input the missing parts
Else IF (the anti-pattern is same name)
THEN
change the elements that have the same name
else if (The anti-patterns are inconsistency anti-patterns)
THEN
see the result of reasoner detection and correct
End

7. Conclusion and Future Work

The proposed method evaluates the quality of UML patterns. The two main distinctions that differ our proposed method from others are; first is that we made the integration between more than one detection tool, which are (ontology, WordNet, OntoUml and Reasoner) to detect the semantic anti-patterns, second, the correction of anti-patterns is automatic in some of anti-patterns
We applied the proposed method on a sample of ten UML class diagrams, which are uploaded as UML templates to be used as patterns in several online references. It detected thirteen semantic anti-patterns. In addition, it detected three syntactical anti-patterns. The "Semantic anti patterns of class" is the most commonly detected anti-patterns group, while the "Semantic anti-patterns of operations" is the least commonly appeared anti-patterns group. The total number of appeared anti-patterns in the sample of ten models is 442.
The main distinction that differ our proposed catalogue from others is the semantic information retrieval, which avoid redundancies in meaning.
In the future, we are going to make the anti-patterns fully automatic corrected. In addition, the highest result will be produced when we create a plugin for OWL in Rodin or create a plugin for Event-B in Protégé to detect both structure and semantic anti-patterns in one work. Finally, we will move to the next step by a general method that can detect both code and design anti-patterns levels.

References

[1] M. E. Elaasar, L.C. Briand, Y. Labiche, PhD thesis: An Approach to Design Pattern and Anti-Pattern Detection in MOF-Based Modeling Languages. PhD thesis, Carleton University, (2012).
[2] G. Guizzardi, & Sales: Detection, Simulation and Elimination of Semantic Anti-patterns in Ontology-Driven Conceptual Models, Science. Vol. 8824, pp 363-376, Springer International Publishing, (2014).
[3] S.R. Idate and N.I. Dalvi. Method to Detect Inconsistencies from Class and Sequence Diagrams, International Journal of Science,

Engineering and Technology Research (IJSETR), Vol.3, Issue 8, (2014).

[4] E.K. Elsayed. Converting UML class diagram with anti-pattern problems into verified code relying on event-b. AIML journal, ISSN 1687-4846, Volume 14, issue 1, ICGST LLC, USA, (2014).

[5] Collection of anti-patterns available at https://sites.google.com/site/metamodelinganti-patterns/catalogueue. Access at (Oct. 2016).

[6] Collection of anti-patterns available at http://c2.com/cgi/wiki?Anti-patternsCatalogueue.Access at (Oct. 2016).

[7] Collection of anti-patterns available at https://en.wikipedia.org/wiki/Anti-pattern.Access at (Oct. 2016).

[8] N. Guarino, and R. Poli. Toward principles for the design of ontologies used for knowledge sharing. In Formal Ontology in Conceptual Analysis and Knowledge Representation, Kluwer Academic Publishers, in press. A substantial revision of paper presented at the International Workshop on Formal Ontology, (1993).

[9] Semagix.com. Advantage of Semantic data, available at http://www.semagix.com/advantage-of-semantic-data.htm, (2009).

[10] X. Jiang & A. H. Tan. Learning and inferencing in user ontology for personalized Semantic Web search, Information sciences, 179 (16), 2794-2808, (2009).

[11] Protégé Platform available at http://protege.stanford.edu/, (2015).

[12] ATM class diagram available at https://www.lucidchart.com/pages/class-diagram-for-ATM-system-UML, (2016).

[13] Library and Android class diagrams available at http://www.uml-diagrams.org/class-diagrams-overview.html, (2016).

[14] Hasp class diagram available at http://www.uml-diagrams.org/software-licensing-class-diagram-example.html?context=cls-examples, (2016).

[15] Seminar, Order and Auction class diagrams available at http://creately.com/diagram/example/gsxncbybt/Seminar+Class+Diagram, (2016).

[16] SWT class diagram available at http://www.ibm.com/developerworks/library/wa-eclipsemvc/, (2016).

[17] Furniture class diagram available at https://www.gliffy.com/go/html5/launch?app=1b5094b0-6042-11e2-bcfd-0800200c9a66&templateId=4218693, (2016).

[18] R. Fourati., N. Bouassida, and H. B. Abdallah. A metric-based approach for anti-pattern detection in uml designs". In Computer and Information Science, (pp. 17-33). Springer, Berlin, Heidelberg, (2011).

[19] OntoUML lightweight editor available at https://code.google.com/p/ontouml-lightweight-editor/, (2016).

[20] Development Anti pattern Road Map available at http://c2.com/cgi/wiki?DevelopmentAntiPatternRoadMap.

[21] Rodin platform available at https://sourceforge.net/projects/Rodin-b-harp/files/Plugin_%20UML-B.

Biographies

Kamal Abdelraouf ElDahshan is a professor of Computer Science and Information Systems at Al-Azhar University in Cairo, Egypt. An Egyptian national and graduate of Cairo University, he obtained his doctoral degree from the Universit de Technologie de Compigne in France , where he also taught for several years. During his extended stay in France, he also worked at the prestigious Institute National de Tlcommunications in Paris. Professor ElDahshan's extensive international research, teaching, and consulting experiences have spanned four continents and include academic institutions as well as government and private organizations. He taught at VirginiaTech as a visiting professor; he was a Consultant to the Egyptian Cabinet Information and Decision Support Center (IDSC); and he was a senior advisor to the Ministry of Education and Deputy Director of the National Technology Development Center. Professor ElDahshan is a professional Fellow on Open Educational Resources as recognized by the United States Department of State.

Asoc. Prof. Eman K. Elsayed, PhD computer science 2005, Alazhar university, Master of computer science, Cairo University 1999, Bachelor of Science, mathematics and computer science Department, Cairo University 1994. She Published thirty three papers until 2016 in data mining, Ontology engineering, e-learning, image processing and software engineering. I also published two books in formal methods and event B on Amazon database. I am a member in Egyptian mathematical society and Intelligent computer and information systems society.

Dr Enas E. El-Sharawy, Computer Science M.Sc 2011 and PHD 2014 from Al-Azhar University, She works as lecturer of computer science at Al-Azhar University. She published many papers until 2014 in Formal method, Rodin platform and UML and interested in database and ontology.

Naglaa. E. Ghannam, Bachelors of Science, mathematics and computer science Department, Al-Azhar University 2002. Currently, She is a master student and working at Al-Azher University.